Module File Cache for NonStop SQL/MX

Technical white paper

Table of contents

Introduction 2
Direct statements 2
Reasons for an SQL Statement cache 2
Re-using statements 2
Organization of the SQL/MX driver statement cache 3
Memory management of the statement cache 4
Statement cache in the JDBC Type 2 driver 4
Statement cache in the JDBC Type 4 driver 5
Statement cache for NonStop ODBC/MX applications 6
Extending the statement cache with a persistent cache 6
Module File Cache 6
Compiling SQL/MX modules 6
Using modules when executing SQL statements 7
How modules are created 9
Advantages of MFC with the MEASURE SQLSTMT entity 11
Measuring the T2 driver 11
Measuring the T4 and ODBC drivers via MXCS 13
Relating MEASURE SQLSTMT entries to SQL statement text 13
Obtaining execution plans 14
Adjusting queries without modifying an application 16
A word of caution 16
Best practices 17
Enable in production environments 17
Use separate module directories per application 17
Share module directories among T2 application instances 17
Placement of the module directory 17
Controlling change: Module management 17
In MXCS, designate specific data sources to MFC 18
Current limitations of MFC 19
Use of extended precision numeric fields 19
Control Directives 19
Calls to Stored Procedures 19
Rowset operations 19
Conclusion 20
References 20
Appendix: The mxmfc utility script 21

Introduction

The Module File Cache (MFC) is an additional, persistent, SQL statement cache that can be used by Java programs using JDBC or other programs that use ODBC[1] to access NonStop SQL databases using the NonStop SQL/MX engine. MFC has been documented in manuals that are targeted to JDBC users, manuals that are targeted to ODBC users, as well as in the *SQL/MX Connectivity Service Manual*. This technical white paper is designed to bring all the information together in one place.

In this paper, the terms SQL query and SQL statement are used to describe any SQL statement: queries that use select, as well as inserts, updates, and deletes.

Direct statements

This paper describes the use of MFC for prepared statements only. Future releases of MFC will support directly executed statements. It is yet undetermined how users can control this support. The option will be described in a later revision of this white paper.

Reasons for an SQL Statement cache

A statement that is submitted by a program to NonStop SQL, needs to be compiled first to create a query execution plan. This compilation step is performed by the SQL/MX compiler (mxcmp). The compiler uses the statement predicates, database statistics and the physical database layout to produce the most efficient plan to execute the request.

The SQL/MX compiler optimizes its compile time by maintaining a cache of statements that it has compiled. This cache is called the QUERY_CACHE in the SQL/MX Reference manual and the SQL/MX Query Guide. However, there are restrictions on the type of statements the compiler is able to cache internally. *Chapter 6 of the SQL/MX Query Guide* describes which statements are supported by this query cache.

Because of these restrictions, MFC was designed to cache in principle every statement that is prepared via the Java JDBC drivers and the Windows® ODBC driver. A few exceptions are described in the section called *Current limitations of MFC*.

Re-using statements

Compilation of a statement can be an expensive operation when comparing it to executing a simple query that returns a single row using a unique key. It is therefore important to re-use statements where possible. In order to re-use statements, the program should prepare (compile) the statements once using parameters for input variables, and execute these statements many times using different parameter values.

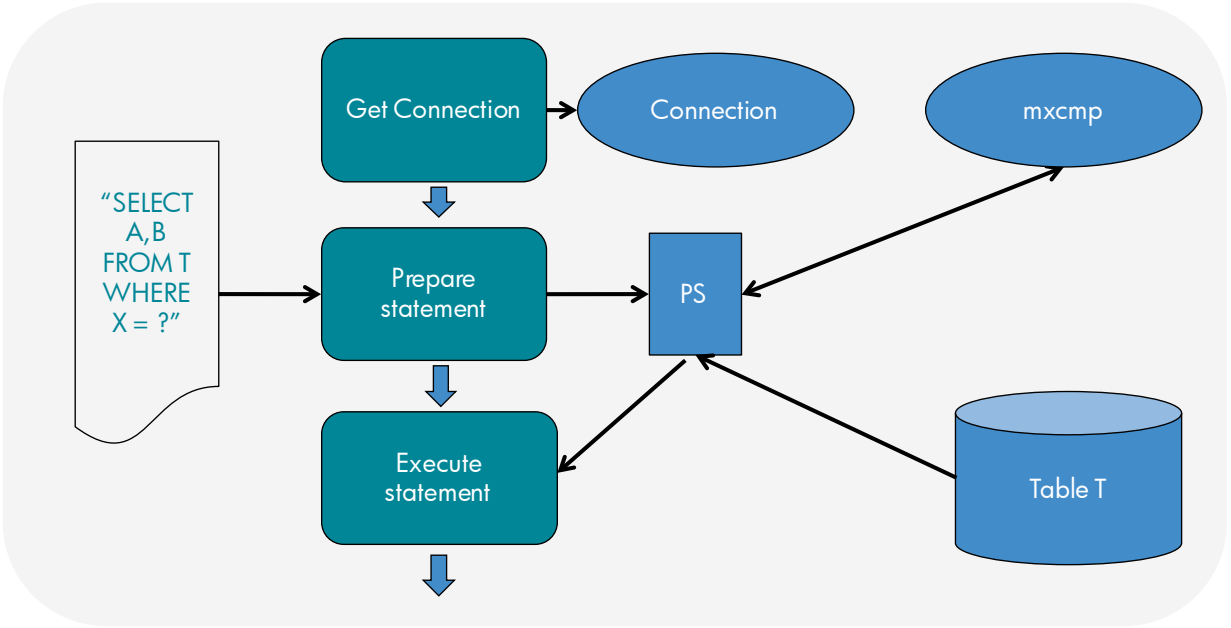
When driver statement caching is in effect, SQL/MX will check if a statement is already in cache prior to invoking the SQL/MX compiler. This allows programs to use a common pattern that contains preparation of the statement and execution of the statement in all cases, without having to manage their own collection of prepared statements and the associated memory.

[1] Currently, MFC support for ODBC clients is limited to clients running on Microsoft® Windows.

Organization of the SQL/MX driver statement cache

The SQL/MX driver statement cache is organized per database connection. This is similar to how other vendors such as Oracle RDBMS, Oracle WebLogic, Apache Derby and others have implemented this. Figure 1 shows how a database connection is obtained, and how a statement PS is prepared and executed. The example shows how the JDBC drivers minimize compilation activity using a statement cache.

Figure 1: Statement execution flow



The prepare step will invoke the mxcmp process if the statement is not yet present in the cache for this connection. Once compiled, the statement will remain in the cache until the connection is destroyed. If connections are pooled by the driver, just closing the connection does not result in removal of all the prepared statements. Connection pooling is a technique used to minimize overhead in obtaining and releasing connections by returning free connections to a pool. The JDBC/MX drivers provide connection pooling when the ***maxPoolSize*** property is set to a number of connections greater than 0.

Applications that use the SQL/MX ODBC driver for Windows can use a similar technique; however, the driver does not maintain a statement cache. The program can create a list of handles, one for each prepared statement, which can act as a statement cache for the duration of the connection. The ODBC driver keeps a connection active for 60 seconds by default and this default can be changed by either modifying the Windows registry or by calling the **SQLConfigDriver** function. ODBC Clients must explicitly enable connection pooling before creating a connection. *(Refer to: Enabling Connection Pooling in the ODBC/MX Driver for Windows manual).*

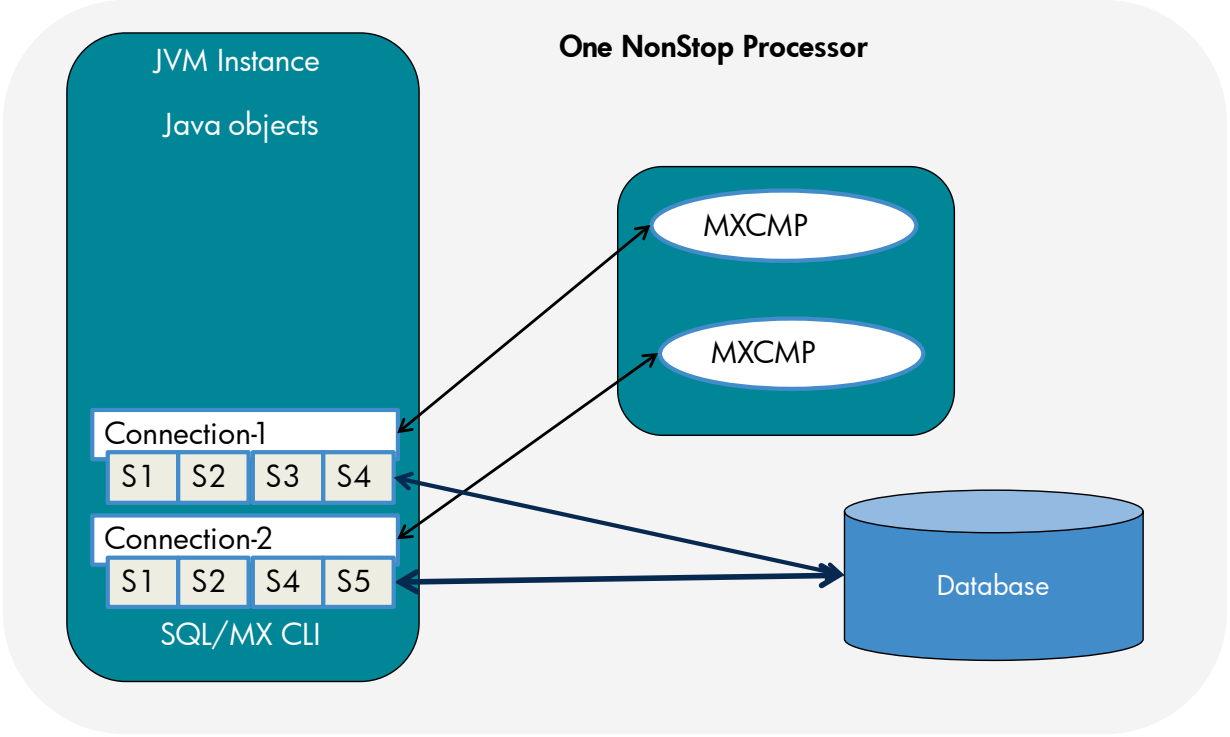
Memory management of the statement cache

The compiled statements and their data structures occupy memory, and memory is a limited resource. Therefore, the size of the cache is configurable by setting the ***maxStatements*** property in Java prior to creating connections. This property is often set as a Java system property as part of the Java startup command. However, it can also be set programmatically by passing the property when connecting to the database. Because the memory requirements of a statement depend on many factors, such as the text of the statements and the execution plan, tuning the required memory is difficult. The JDBC/MX drivers free memory of statements that are not in use, based on a Least Recently Used (LRU) algorithm.

Statement cache in the JDBC Type 2 driver

The JDBC Type 2 (T2) driver is used by Java programs, such as NonStop Servlets for Java Server Pages (NSJSP) servers, TS/MP servers written in Java, SQL/MX Stored Procedures or standalone Java programs that are running on the NonStop server. The driver interfaces with the SQL/MX Call Level Interface (CLI) directly and the driver operates under the same user-ID as the Java Virtual Machine (JVM) process. The advantage of using the T2 driver is in the shorter path length to the database. A possible downside is that the available process memory must be shared by the JVM, the SQL/MX executor code, buffer space, and the statement cache(s) for the connections. Figure 2 shows how a single JVM with two connections uses the statement cache and interfaces to the compilers and the database.

Figure 2: Java VM using the T2 driver



The compilers, one for each connection, are automatically started when a new connection is created and they run in the same logical processor as the JVM. Java processing and statement compilation compete for the same processor resources. However, data access can be to any disk in the cluster thanks to the distributed nature of the NonStop SQL database.

Statement cache in the JDBC Type 4 driver

The JDBC Type 4 (T4) driver is used by Java programs that are not running on the NonStop server, such as Java application containers like Oracle WebLogic, IBM WebSphere, and Red Hat JBoss which can all run on Microsoft Windows, and various types of UNIX® and Linux environments. In addition, customers may choose to use the T4 driver even in applications that do run on the NonStop server, thus spreading the load over multiple logical processors.

The T4 driver is a 100 percent pure Java driver and it connects to the SQL/MX Connectivity Service (MXCS) using TCP/IP.

Figure 3: Java VM using the T4 Driver

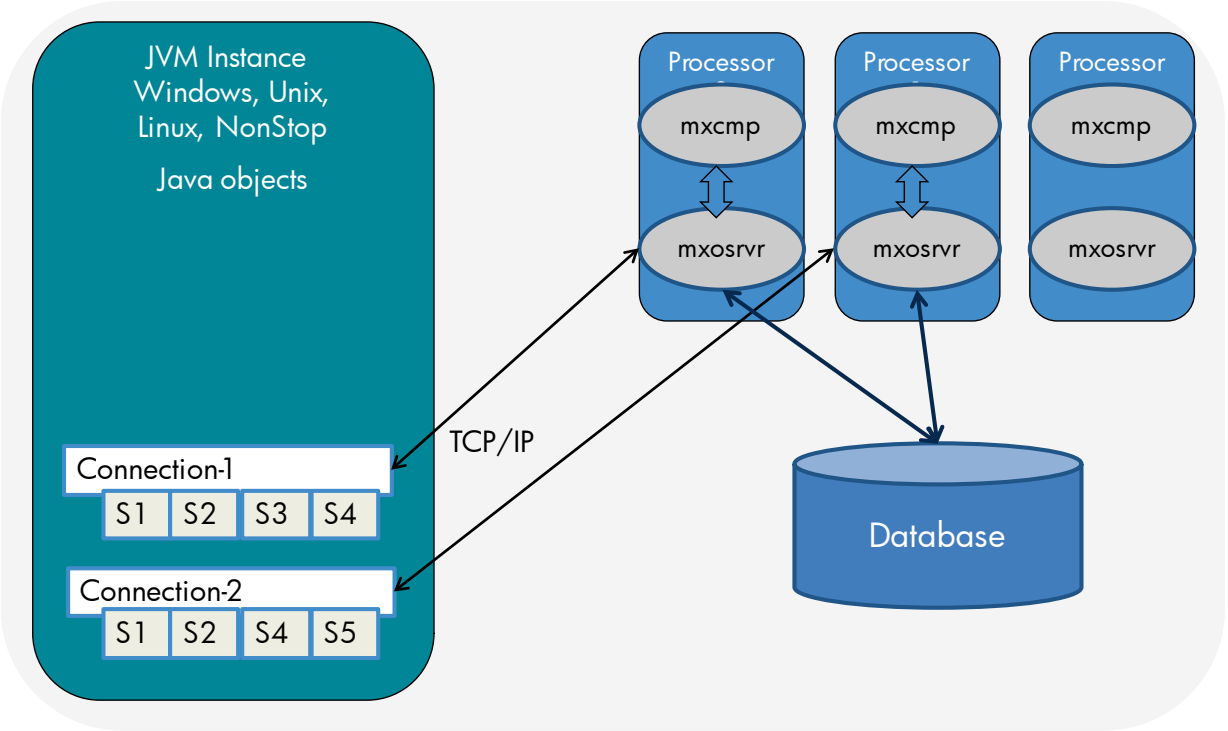


Figure 3 shows how a JVM uses two connections to MXCS via TCP/IP. JVMs that are running on the NonStop server and use the T4 driver also connect to MXCS using TCP/IP. MXCS is a subsystem that represents the database as one or more data sources to the outside world. More information about the MXCS architecture can be found in *Chapter 1 of the SQL/MX Connectivity Service Manual*.

Each connection uses a single-threaded mxosrvr process which acts as a proxy for the connection object that is present in the JVM. The mxosrvr process interfaces with the SQL/MX compiler (mxcmp) to create query execution plans and interfaces with the CLI in its process space to execute these plans. The cached query plans therefore reside in the address space of the mxosrvr process, not in that of the JVM, as is the case for a T2 driver. A statement cache in the JVM exists; however, it contains only a reference to the cached statements in mxosrvr.

With the T4 driver, the compiler activity of an application as well as the database access activity is distributed over multiple processors, which may be an advantage over a single JVM using multiple connections and the T2 driver. The second advantage is that the JVM does not have to share its memory address space with the SQL/MX CLI and statement cache.

A possible downside is that MXCS needs to be started and that the connection requires a valid user-ID and password to login. The T2 driver does not require a login and the simple fact that it is running, means that the system is also running and the database is available.

Statement cache for NonStop ODBC/MX applications

ODBC/MX client programs running on Microsoft Windows, HP NonStop servers or other platforms connect to the same MXCS infrastructure as the Java T4 clients. There is no automatic caching as provided by the Java drivers. Customers can execute their statements but need to perform their own memory management of the prepared statements.

However, the mxosrvr processes will keep prepared statements in memory if the client uses different statement handles until the client disconnects.

Extending the statement cache with a persistent cache

The in-memory statement cache is maintained per database connection, and statements need to be compiled by the SQL/MX compiler before they can be stored in the memory cache. This means that when an application uses 100 unique statements and it uses 100 connections, 10,000 compiles need to take place before all statements have been placed in the in-memory cache. For this reason, many customers begin with preparing all the SQL statements when the application is started, so that the transaction path will not include a compilation step. However, this means that the startup time of an application will be proportional to the number of connections and the number of statements.

If the total amount of memory is not enough to store the compiled execution plans, SQL/MX frees memory by removing older statements and their plans from the memory cache. If such a removed statement is used again by an application, another compilation needs to take place.

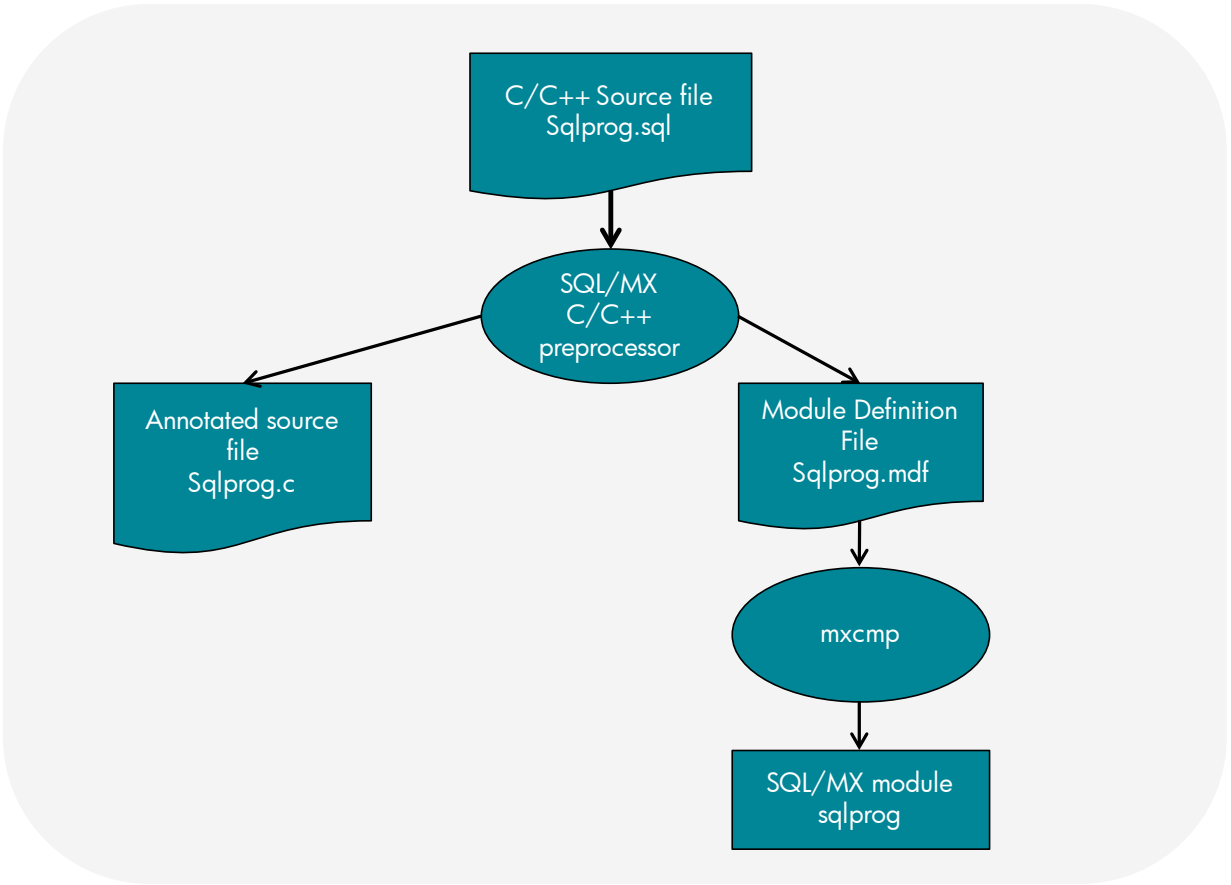
Adding a persistent statement cache, available to all connections, solves these two issues by allowing SQL/MX to read already prepared execution plans from disk instead of having them compiled. Instead of 10,000 compilations, only 100 are needed. The persistent statement cache is implemented by creating SQL/MX compiled modules and this is the reason the feature is called Module File Cache (MFC). The next sections describe MFC in detail.

Module File Cache

Compiling SQL/MX modules

MFC uses some of the building blocks that are available in SQL/MX to support embedded SQL in C/C++. SQL statements and data declarations that are developed in these languages are expressed in a specific syntax. A separate preprocessor, called mxsqlc, translates the declarations and statements into C/C++ data declarations and calls to the CLI. In addition, the preprocessor creates a module definition file (MDF). This module definition file is input to the SQL/MX compiler mxcmp, which uses the statements in the MDF to create a module that contains the query execution plans for the statements. Figure 4 shows how a module is created by executing the preprocessor and the SQL/MX compiler. Figure 4 is a simplified version of figure 15-2 in the *SQL/MX 3.1 Programming Manual for C and COBOL*.

Figure 4: Compilation from source to module

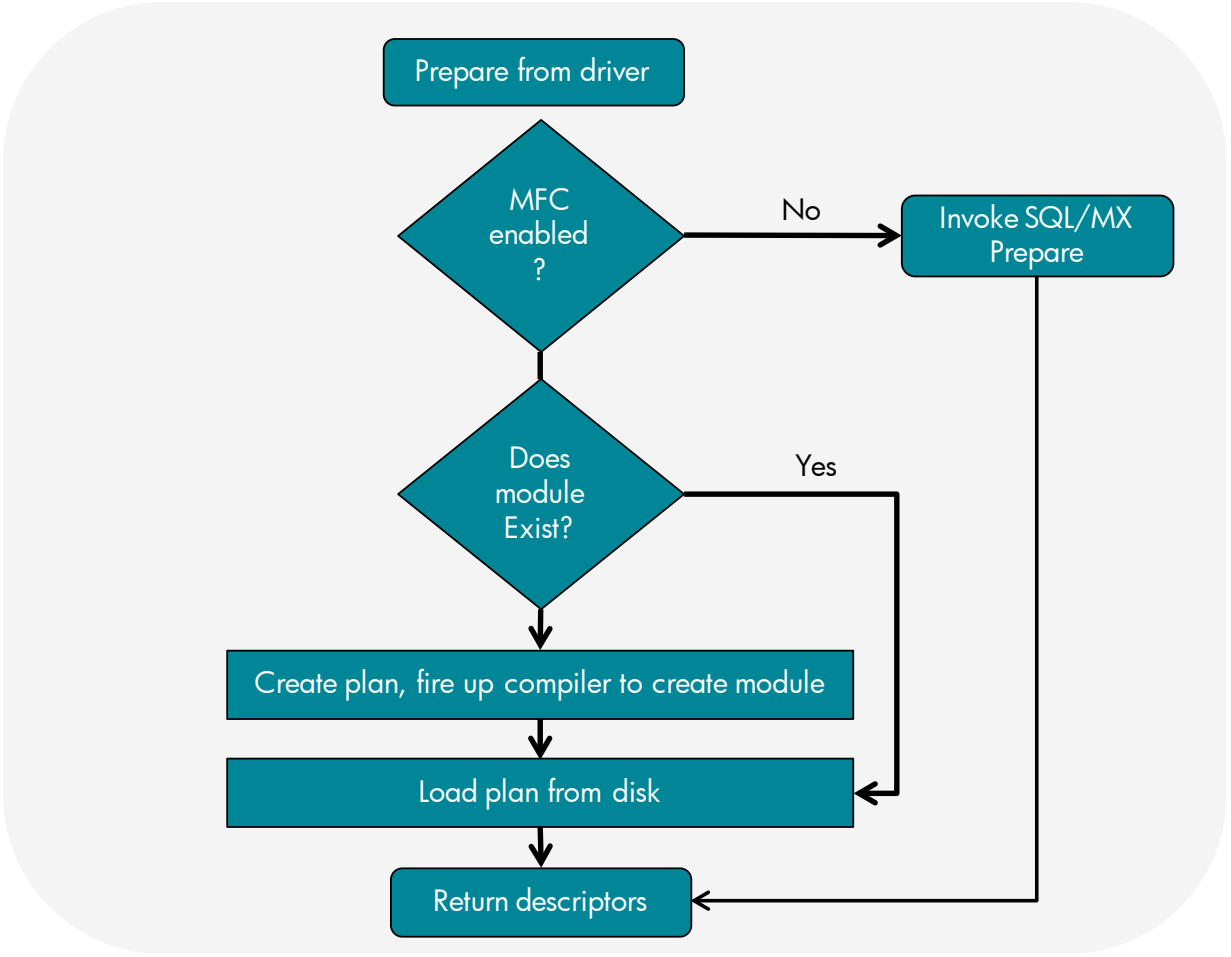


Using modules when executing SQL statements

When a Java application invokes the Connection.prepareStatement() method or when an ODBC/MX application invokes the SQLPrepare() function, the JDBC T2 driver or the mxosrvr process will first check if the statement is in the in-memory cache. If it is not found there, it executes the steps as pictured in Figure 5.

When a module needs to be created, SQL/MX will first create a source file, and then call the preprocessor and the compiler; these are only started when needed, unlike the mxcmp instance that is always present for each connection. Temporary files that were created as part of the module generation step are removed, but the output of the C compiler (the .lst file) and the .mdf file will remain in the module directory to facilitate debugging in case modules fail to be created. The mxmfc management script, which is discussed later in *Module management*, can be used to cleanup these temporary files.

Figure 5: The steps to create a module



When modules source files are created, a .lock file prevents multiple instances from creating the same source at the same time. When SQL/MX looks for a module and finds a .lock file, it will obtain a query execution plan from the compiler instead of waiting for the module to appear. If a module creation should fail, the .lock file prevents SQL/MX from attempting to create modules over and over.

Equality of statements

The driver determines that a statement has already been compiled using the statement text, the current value of schema and catalog and the active Compiler Control Statements,[2] because these controls influence the plan generation. Note that upper case characters, tab characters, and so on are relevant for the equality, even though the SQL language is not case-dependent. The reason for this restriction is mainly for performance reasons; the driver cannot look at the details of the SQL statements because it does not include an SQL language parser and adding one is not very cost-effective. Secondly, since MFC is used by applications, it is expected that the same statement will be prepared multiple times using the same text.

[2] These control statements are Control Query Defaults, as well as Control Table statements.

How modules are created

Module creation and usage is optional and can be defined at the level of the JVM for the T2 driver and at the level of the MXCS Data Source for T4 and ODBC/MX drivers. This is because the T2 driver is responsible for creating the module source on the NonStop file system[3] and subsequently to invoke the preprocessor and mxcmp processes. In MXCS, for T4 and ODBC applications, this is the responsibility of the mxosrvr process.

Module naming convention

The catalog name, the schema name, the Active Controls and the statement text together are hashed into a string value of 32 bytes. This string is then preceded by catalog and schema, and some additional data.

The following module names illustrate the convention:

`FRANS_31.PERF.T2MFC000670940A89478A2B397FB597FED24D`
A module created by the T2 driver contains: Schema "FRANS_31.PERF", driver tag: "T2MFC", and the hash value.

`MFCABS1031FRANS_31.PERF.087457CC9FB94B6A2A03F3FFC9784479`
A module created by MXCS contains: Driver tag: "MFCABS1031", Schema "FRANS_31.PERF", and the hash value.

Example of generated module source

The SQL/MX controls and schema settings are copied into the generated source code. Figure 6 shows an example of a generated source file. It contains the statement string that was prepared by the programs as a C-language comment.[4] A table control that defines the timeout value was set explicitly by the program and it was saved in the generated module because it needs to be used by the SQL compiler to produce a plan that includes this setting. Note that the SELECT statement does not need to include the catalog and schema names. The name of the active schema is included as DECLARE and SET statements. Applications can be developed independently from catalog and schema which makes a single-source code for different environments possible.

Figure 6: Generated .sql file

```
#select * from tab1
# include<stdio.h>
EXEC SQL MODULE FRANS_31.PERF.T2MFCC01514F420A244CCF176076F91521E16 NAMES ARE
ISO88591;
int main ()
{
 EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION  ;
EXEC SQL CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE';
EXEC SQL CONTROL QUERY DEFAULT RECOMPILATION_WARNINGS 'ON';
EXEC SQL control table * timeout  '10';
EXEC SQL DECLARE SCHEMA FRANS_31.PERF ;
EXEC SQL SET SCHEMA FRANS_31.PERF ;
EXEC SQL DECLARE MXSTMT01 CURSOR FOR select * from tab1 ;
return 0;
 }
```

[3] All files are created in the OSS environment.
[4] In post SQL/MX 3.1 releases the statement will be commented in another way.

Enabling MFC for the T2 driver

The T2 driver uses the following properties to control MFC. It is important to understand that MFC will only be in effect if statement caching and connection pooling are also used:[5]

`jdbcmx.maxPoolSize=<max_number_of_connections>`

This property defines the maximum number of connections to the database. If a new connection is created the first time, SQL/MX will launch an mxcmp process that belongs to that new connection. When the application closes a connection, the driver will only soft-close the connection but keep processes and data structures, including the cached statements in place. A related property—minPoolSize—can be used to keep a lower number of connections alive than set by maxPoolSize.

`jdbcmx.maxStatements=<max number of statement in memory cache>`

This property defines the maximum number of statements per connection in the in-memory cache. If there is not enough room in memory, SQL/MX will free memory by removing the least recently used statements.

`jdbcmx.enableMFC=ON`

This option instructs the JVM to create modules and read them from disk when needed.

`jdbcmx.compiledModuleLocation=<full OSS directory name>`

The compiledModuleLocation defines the directory name where the driver will store the modules and temporary files. For example: /home/frans/invent/modules. When the JVM starts, the directory must exist, otherwise the JVM will abend.

The properties are usually set as a system property when the JVM is started, but a program may also set them after the JVM has been started. HP does not recommend using different values of the properties for different connections; results may be unpredictable.

Enabling MFC for the T4 and ODBC Drivers

Module creation and use is done by the mxosrvr processes which are configured and controlled by the MXCS subsystem. MXCS is managed by NSM/Web—a web-GUI tool—and by the MXCS control language supported by mxci, the SQL/MX command line interface. The MXCS configuration can be done with the following SET commands in mxci:

```
SET STATEMENT_MODULE_CACHING 'TRUE'
```

In mxci, the SET command has its own syntax and includes the name of the MXCS data source that it applies to. For example, to enable MFC for data source INVENT, the following mxci commands would be used:

```
>> MODE MXCS;
CS> ADD EVAR $MXOAS.INVENT.STATEMENT_MODULE_CACHING, TYPE SET , VALUE 'TRUE';
```

The first line invokes MXCS mode. The second line, sets the option, called an EVAR, and it includes the MXCS service—$MXOAS, the data source within the service (this is what JDBC calls the "serverDataSource"), and the control option and value.

```
SET COMPILED_MODULE_LOCATION <full OSS directory name>
```

This is an example to set the module directory for the designated data source. It shows how the module directory is defined as /home/frans/invent/mxcsModules.

```
CS> ADD EVAR $MXOAS.INVENT.COMPILED_MODULE_LOCATION, TYPE SET, VALUE
'/home/frans/invent/mxcsModules';
```

It is possible to share one directory between the T2 and MXCS generated modules. The names of the modules created by the T2 driver have a different structure than those generated by MXCS. When multiple data sources serve the same application, it is recommended to configure the same directory for those data sources to reduce compilations.

Explicitly turning MFC OFF for T4 applications

MFC is enabled at the MXCS data source level. This means that any application that connects to such a data source, will automatically create required modules when a statement is prepared and that modules will be retrieved from disk as needed. There may be situations where you do not want to use MFC; for example, if you want to make sure that the compiler is invoked every time when a statement is prepared.[6] In these cases, the T4 driver property t4jdbc.enableMFC should be set to OFF.

---

[5] This is true in releases including SQL/MX 3.1. In future, this restriction may be lifted.

[6] Examples of such tools are Visual Query analyst (VQA) and Visual Query Planner (VQP), that are designed to provide the user insight in query execution plans, based on current statistics.

Advantages of MFC with the MEASURE SQLSTMT entity

Many customers are used to using the Measure tool for performance tuning their applications. When dynamic SQL is used (as opposed to embedded SQL in COBOL or C/C++ programs) it is difficult to find the cause of poor performance, because the SQL statement text is not available to the person performing the analysis.

The following examples use the following queries.

```
select count(*) total_rows from T01;

select count(*) col35a_count from t01 where col35a <> ' ';
```

In the examples, the queries are executed two times with statement caching enabled. This means that the Measure counters such as: number of calls, the number of records accessed/used are the sum of the two executions.

Measuring the T2 driver

When statements are prepared and executed, the Measure tool shows the statement activity shown in the figure 7. An SQL Statement Procedure is identified by a name that is internal to the driver.

Figure 7: MEASURE SQLSTMT entity

```
SQL Statement
Procedure 'MXID010780307412121877008255031800000000000210HP.FJONGMA00_5_STMT1'
Process  3,741                Pri 149        OSSPID:  122552387
Program  $OSS.ZYQ00000.Z0000DWM:84417085
OSSPath: "/usr/tandem/nssjava/jdk160_h60/bin/java"
Userid   211,96    Creatorid  211,96    Ancestor 1,336
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  10 Nov 2011,  8:00:32   For  34.7 Seconds
------------------------------------------------------------------------
Elapsed-Busy-Time           46.43 ms   Calls                          4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      12 #     Message-Bytes             81,344 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                 3,986 #
Records-Accessed         8,000,000 #   Records-Used           8,000,000 #
++
SQL Statement
Procedure 'MXID010780307412121877008255031800000000000210HP.FJONGMA00_6_STMT2'
Index #0
Process  3,741                Pri 149        OSSPID:  122552387
Program  $OSS.ZYQ00000.Z0000DWM:84417085
OSSPath: "/usr/tandem/nssjava/jdk160_h60/bin/java"
Userid   211,96    Creatorid  211,96    Ancestor 1,336
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  10 Nov 2011,  8:00:36   For  31.1 Seconds
------------------------------------------------------------------------
Elapsed-Busy-Time           19.12 ms   Calls                          4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      15 #     Message-Bytes             93,504 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                16,345 #
Records-Accessed         8,000,000 #   Records-Used           1,729,958 #
5+
```

The first statement is the count of all rows, the second statement is the counting with the WHERE clause (because it shows less rows used than the number that are accessed). When many statements are measured, which will be the case in a "real-life" environment, it is impossible to find which statement is responsible for the resource usage.

When MFC is enabled and modules are retrieved from disk,[7] statements are traceable and the Measure output resembles what is displayed in Figure 8. If a module is used, the SQL Statement Procedure entry points at the location of the module file on disk and this means that it can be examined for performance analysis. The two SQLSTMT entries below show the modules that were used to execute the two statements. Note how the module names contain the catalog and schema names, as well as the T2 MFC tag.

Figure 8: MEASURE SQLSTMT entity with MFC enabled

```
SQL Statement
Procedure '/home/frans/modules/FRANS.PERF.T2MFC843127986EDAC1ED6C96A4D505523EFB'
Index #4
Process  3,781                Pri 149        OSSPID:  323878979
Program  $OSS.ZYQ00000.Z0000DWM:84417085
OSSPath: "/usr/tandem/nssjava/jdk160_h60/bin/java"
Userid   211,96    Creatorid  211,96    Ancestor 1,336
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  10 Nov 2011,  8:32:46   For  34.5 Seconds
------------------------------------------------------------------------
Elapsed-Busy-Time           63.99 ms   Calls                          4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      12 #     Message-Bytes             81,440 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                 3,986 #
Records-Accessed        8,000,000 #    Records-Used           8,000,000 #
++
SQL Statement
Procedure '/home/frans/modules/FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376'
Index #4
Process  3,781                Pri 149        OSSPID:  323878979
Program  $OSS.ZYQ00000.Z0000DWM:84417085
OSSPath: "/usr/tandem/nssjava/jdk160_h60/bin/java"
Userid   211,96    Creatorid  211,96    Ancestor 1,336
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  10 Nov 2011,  8:32:50   For    31 Seconds
------------------------------------------------------------------------
Elapsed-Busy-Time           19.54 ms   Calls                          4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      15 #     Message-Bytes             93,504 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                16,345 #
Records-Accessed        8,000,000 #    Records-Used           1,729,958 #
4+
```

[7] However, the instance that creates a module on disk, Measures the statement by its internal name.

Measuring the T4 and ODBC drivers via MXCS

All data access for the JDBC T4 and ODBC/MX drivers is performed via the mxosrvr processes that are controlled by the MXCS subsystem.

When the statements are retrieved from the modules instead of being recompiled, they appear in MEASURE similar to the entities from the T2 driver. The example below shows the same statements executed via the T4 driver, and therefore the SQLSTMT entities belong to an mxosrvr process. Note the driver tag (MFCABS1031), the default catalog, and the schema in the module file name.

Figure 9: MEASURE SQLSTMT entity for mxosrvr in MXCS

```
SQL Statement
Procedure '/home/frans/modules/MFCABS1031FRANS.PERF.0ED8648131C54AC3578D42CDE8E2C3D5'
Index #9
Process  2,947     ($Z1HC)     Pri 168        OSSPID:  357433414
Program  $SYSTEM.ZMXODBC.MXOSRVR
OSSPath: "/G/system/zmxodbc/mxosrvr"
Userid   211,96    Creatorid  211,96    Ancestor 2,631 ($MXOAS)
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  11 Nov 2011,  5:45:46   For  38.2 Seconds
------------------------------------------------------------------------------
Elapsed-Busy-Time            6.73 sec  Calls                           4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      12 #     Message-Bytes              81,440 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                  3,986 #
Records-Accessed        8,000,000 #    Records-Used            8,000,000 #
++
SQL Statement
Procedure '/home/frans/modules/MFCABS1031FRANS.PERF.BF04D0FD6C8104F5B5C8295BC3913B39'
Index #9
Process  2,947     ($Z1HC)     Pri 168        OSSPID:  357433414
Program  $SYSTEM.ZMXODBC.MXOSRVR
OSSPath: "/G/system/zmxodbc/mxosrvr"
Userid   211,96    Creatorid  211,96    Ancestor 2,631 ($MXOAS)
Format Version:  H06  Data Version:  H06  Subsystem Version:  1
Local System \NSKIT11 From  11 Nov 2011,  5:45:49   For  34.7 Seconds
------------------------------------------------------------------------------
Elapsed-Busy-Time           26.90 sec  Calls                           4 #
Elapsed-Sort-Time                      Sorts
Elapsed-Recompile-Time                 Recompiles
Messages                      15 #     Message-Bytes              93,504 #
Lock-Waits                             Escalations
Timeouts                               Disc-Reads                 16,339 #
Records-Accessed        8,000,000 #    Records-Used            1,729,958 #
5+
```

Relating MEASURE SQLSTMT entries to SQL statement text

To find the statement text for a module that is present in the MEASURE SQLSTMT entity, the accompanying SQL text file must be examined. For example, if you want to know which statement accesses 8,000,000 rows and uses only 1,729,958, simply list the text using the cat command in OSS. The name of the SQL text file is the same as the module with a .sql suffix added. The next example shows the text reported by measuring the T2 driver in Figure 8. The reason why many records are accessed, but not are all used, is because the query needs to examine all the rows but not all will qualify the search criterion.

Figure 10: Contents of the MFC generated SQL file

```
~> cat /home/frans/modules/FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376.sql
#select count(*) col35a_count from t01 where col35a <> ' '

# include<stdio.h>

EXEC SQL MODULE FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376 NAMES ARE ISO88591;
int main ()
{
 EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION  ;
EXEC SQL CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE';
EXEC SQL CONTROL QUERY DEFAULT RECOMPILATION_WARNINGS 'ON';
EXEC SQL DECLARE SCHEMA FRANS.PERF ;
EXEC SQL SET SCHEMA FRANS.PERF ;
EXEC SQL DECLARE MXSTMT01 CURSOR FOR select count(*) col35a_count from t01 where
col35a <> ' ' ;
return 0;
 }
~>
```

Using mxmfc described in the Appendix, the statement text can be retrieved using the module name that MEASURE reports as follows:

Figure 11: mxmfc output of statement text

```
~> mxmfc -S -m
/home/frans/modules/FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376
#select count(*) col35a_count from t01 where col35a <> ' '
~>
```

Obtaining execution plans

The SQL/MX modules contain the execution plan of the queries. These execution plans can be examined using the VQP and VQA tools as well as with mxci.

Figure 11 shows how the SQL/MX execution plan can be retrieved using the SQL/MX command interface, mxci. The example uses the EXPLAIN command with a wildcard character to select all statements in the module. MFC modules contain only one single SQL DML statement. However, there may be additional statements that are used to preserve the context as it was when the module was generated. The generated SQL SELECT statements will be called "MXSTMT01", but other statements have a different name. The "esp_exchange" operation in the example shows that Executor Server Processes (ESPs) are used to scan 4 partitions in parallel. The statement text can also be found by listing the module definition file (with suffix .mdf) and searching for the index number reported by MEASURE (in the T2 example, index # 4).

Figure 12: Retrieving the SQL execution plan using mxci

```
~> mxci
Hewlett-Packard NonStop(TM) SQL/MX Conversational Interface 3.1
(c) Copyright 2003, 2004-2011 Hewlett-Packard Development Company, LP.
>>EXPLAIN options 'f'  '%' from
'/home/frans/modules/FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376' ;

Statement: MXSTMT01

LC   RC   OP   OPERATOR              OPT       DESCRIPTION           CARD
---- ---- ---- --------------------  --------  --------------------  ---------

6    .    7    root                                                  1.00E+000
5    .    6    sort_partial_aggr_ro                                  1.00E+000
4    .    5    esp_exchange                    1:4(range)            1.00E+000
3    .    4    sort_partial_aggr_no                                  1.00E+000
2    .    3    partition_access                                      1.99E+006
1    .    2    sort_partial_aggr_le                                  1.99E+006
.    .    1    file_scan             fs        T01 (s)               3.99E+006

Statement: SQLMX_DEFAULT_STATEMENT_2

LC   RC   OP   OPERATOR              OPT       DESCRIPTION           CARD
---- ---- ---- --------------------  --------  --------------------  ---------

.    .    1    root                                                  1.00E+000

Statement: SQLMX_DEFAULT_STATEMENT_3

LC   RC   OP   OPERATOR              OPT       DESCRIPTION           CARD
---- ---- ---- --------------------  --------  --------------------  ---------

.    .    1    root                                                  1.00E+000

Statement: SQLMX_DEFAULT_STATEMENT_4

LC   RC   OP   OPERATOR              OPT       DESCRIPTION           CARD
---- ---- ---- --------------------  --------  --------------------  ---------

.    .    1    root                                                  1.00E+000

Statement: SQLMX_DEFAULT_STATEMENT_5

LC   RC   OP   OPERATOR              OPT       DESCRIPTION           CARD
---- ---- ---- --------------------  --------  --------------------  ---------

1    .    2    root                                                  1.00E+000
.    .    1    control                                               1.00E+000

--- SQL operation complete.
>>exit;
```

The mxmfc script also allows the retrieval of the execution plans. The same output as shown above can be obtained with mxmfc with the following command:

```
~> mxmfc -l  -d /home/frans/modules -m
FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376
```

Adjusting queries without modifying an application

Sometimes it may be necessary to add controls to an SQL statement to enhance performance. A typical use case is the addition of a CQD, such as INTERACTIVE_ACCESS, or a lock timeout value. Adding such controls to an application may not always be feasible, for example, if the application is provided by a third party. Having modules of the SQL statement is an advantage, because the controls can be placed in the generated source code and the modules can then be recompiled with the modified control settings. SQL recompilation is done using the SQL/MX preprocessor and compiler and can be executed manually as described in the SQL/MX 3.x Programming Manual for C and COBOL, or by invoking the mxmfc utility script which is described in *the Appendix The mxmfc utility script, on page 21*.

Figure 12 shows an example where a query is changed to use BROWSE ACCESS instead of the default access mode which is STABLE ACCESS. The example also shows that an extra CQD was added.

Figure 13: Modified query access mode

```
~/modules> cat FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376.sql
#select count(*) col35a_count from t01 where col35a <> ' '

# include<stdio.h>

EXEC SQL MODULE FRANS.PERF.T2MFC1E8815514F1819F4EDCE6400A9900376 NAMES ARE ISO88591;
int main ()
{
 EXEC SQL BEGIN DECLARE SECTION;
EXEC SQL END DECLARE SECTION  ;
EXEC SQL CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE';
EXEC SQL CONTROL QUERY DEFAULT RECOMPILATION_WARNINGS 'ON';
exec sql control query default interactive_access 'ON';
EXEC SQL DECLARE SCHEMA FRANS.PERF ;
EXEC SQL SET SCHEMA FRANS.PERF ;
EXEC SQL DECLARE MXSTMT01 CURSOR FOR select count(*) col35a_count from t01 where
col35a <> ' ' BROWSE ACCESS;
return 0;
 }
~/modules>
```

A word of caution

As the example shows, it is possible to change any query without actually changing the application. This is a very powerful feature, but it also means that you need to be careful in doing so. You need to make sure that the query will return the same data fields as the original query. You also need to make sure that the module directory is protected from unauthorized write access.

When generated sources have been manually modified, you must make sure that they do not get removed by accident or overwritten by a regeneration of source code.

Best practices

This chapter contains a set of common/best practices. Select these after validating their usefulness in your environment.

Enable in production environments

Creating modules is an expensive operation when compared to just the compilation of an SQL statement. For this reason, MFC is suited for environments that are relatively stable, such as production environments where over the lifetime of an application, the same statement is prepared many times. In development, statements may be changed frequently as part of the development process.

Use separate module directories per application

Having all modules in the system stored in one place makes management of these modules a difficult task. The names of the modules are generated by SQL/MX, and therefore the names cannot be used for easy identification. When all modules that belong to a single application are located in the same directory, regeneration and recompilation of all statements can be done easily by removing everything from the directory and starting the application.

If changes have been made that require the recompilations, for example, when you have added an index and want queries to use the new index, the modules for a given application can be manually recompiled using the .sql files that are present in the application directory.

The use of the USERMODULES directory (`/usr/tandem/sqlmx/USERMODULES`) is not recommended for the same reason of difficulty of management.

Share module directories among T2 application instances

MXCS groups a number of connections, represented by mxosrvr instances, allowing an application to scale. Applications that use Java on the host and the T2 driver can do the same by grouping a number of JVM instances that serve one application. Examples are applications that use the NSJSP container or TS/MP applications that use servers written in Java. If these JVMs enable MFC, they can share the module directory.

Stored procedures in Java (SPJ)

A special kind of T2 applications is formed by SQL/MX Stored Procedures in Java (SPJ). These SPJs are run in designated JVMs separated from the main application. SPJs can use MFC just like normal Java applications and should share their modules in a designated module directory, for example, one per database schema.

Placement of the module directory

Prior to enabling MFC, make sure that the designated directory exists and is writable to the user that eventually creates the modules. On heavily used module directories, for example applications that use many statements or many connections, place them in the root file set and not in secondary file sets. Secondary file sets require multiple accesses to the OSS name server to find a file. If disk IO to the module directory should become a bottleneck, a designated OSS file set as is described in the *JDBC Type 2 driver programmer's reference and SQL/MX connectivity service manuals*, may be useful.

Controlling change: Module management

When database changes occur, the modules may need to be recompiled if the changes affect the stored execution plans. The following options can be used to keep the modules up-to-date.

Clearing all the module cache

Because the modules can be recreated easily and because all modules of an application are in one directory, one option is to simply remove the modules and recompile them.

Locating specific modules using mxci

Module files can be examined by the mxci command called "DISPLAY USE OF".

For example, assume that you want to drop index T01A01 and T01A03 from an application that uses module directory '/home/frans/modules'. The next examples show how to locate the modules that will be affected by this DDL change and can be determined using mxci. The example shows that index T01A01 is used in two modules, one that has been created by the T2 driver and the other by MXCS.

Figure 14: Using DISPLAY USE OF in mxci

```
>>display use of module_dir '/home/frans/modules' object 'FRANS.PERF.T01A01';

Object: FRANS.PERF.T01A01

  Index: FRANS.PERF.T01A01  Module:
/home/frans/modules/FRANS.PERF.T2MFC843127986EDAC1ED6C96A4D505523EFB
  Index: FRANS.PERF.T01A01  Module:
/home/frans/modules/MFCABS1031FRANS.PERF.0ED8648131C54AC3578D42CDE8E2C3D5
>>cd /home/frans/modules;
>>display use of module_dir '*' object 'FRANS.PERF.T01A03';

*** WARNING[15992] Usage information was not found or could not be displayed.
```

The example shows that T01A03 is not used by any of the modules and can be removed safely. When index T01A01 is removed, the two modules can be recompiled using the mxmfc tool.

Locating specific modules using OSS commands

Locating modules may be quicker if the FLOSS[8] grep command is used as is shown next. References to index files will only appear in the binary modules. *Using grep* looking for the base table, T01 will return many more hits because it is referred to in the source files as well as in the modules.

Figure 15: Using grep to locate use of SQL/MX objects

```
~/modules> /usr/local/bin/grep T01A01 *
Binary file FRANS.PERF.T2MFC843127986EDAC1ED6C96A4D505523EFB matches
Binary file MFCABS1031FRANS.PERF.0ED8648131C54AC3578D42CDE8E2C3D5 matches
~/modules> /usr/local/bin/grep T01A03 *
~/modules>
```

In MXCS, designate specific data sources to MFC

If an application uses MFC, define a separate MXCS data source and module directory for this application. If multiple data sources are defined for the same application, these applications can share their modules.

Do not use MFC on the TDM_Default_DataSource DSN, because this is the one that is selected by MXCS as default if a client specifies a name of a data source that has not been configured.

Users of the ODBC-based Visual Query Planner (VQP) should use a separate data source that has MFC turned off in order to avoid unintentional module creation.

[8] Freeware Look for OSS (FLOSS) is the name of the Open Source tools that can be downloaded from the Connect User Group website at http://ituglib.connect-community.org/ FLOSS grep allows searching binary files in addition to ASCII files.

Current limitations of MFC

The existing documentation of MFC lists various limitations regarding scalar functions.[9] These limitations no longer exist.

Use of extended precision numeric fields

With SQL/MX release 3.0, an extended numeric precision data type was introduced. This data type is currently only supported for the mxci, JDBC, and ODBC interfaces. MFC relies on the embedded SQL infrastructure, and this is currently not able to generate the appropriate host variables that are required for inserting data and or inclusion in the WHERE clause of a statement. However, MFC is able to generate a module for statements that SELECT this data type.

Control Directives

Control directives (CQD) are used to influence the behavior of the compiler when it compiles a statement into an execution plan. Controls that are set to improve one statement may have a bad influence on a statement that is compiled later. Therefore, any control that is set dynamically must also be reset immediately.

CQDs set by a program prior to preparing a statement are not copied into the generated source by connections that are handled by MXCS (JDBC T4 and ODBC/MX).

CQDs that are set by a T2 driver may result in an extra module when a statement is prepared when the CQD is set to "ON" and later set to "OFF". The generated source may contain the two CQDs in the order they were executed.

If control directives are required, HP recommends setting them at the system or data source level, have programs set them when connections are created, or to include them in the generated MFC sources and recompile them into plans separately.

Calls to Stored Procedures

Calls to Stored Procedures do not result in the creation of modules when MFC is used with the T2 driver.

Rowset operations

The HP Connectivity Service manual states that MFC is not supported for rowset operations.

[9] *JDBC Type 4 Driver Programmer's reference for SQL/MX Release 3.1; HP NonStop SQL/MX Connectivity Service manual for Release 3.1*

Conclusion

Module File Caching has advantages in many aspects. It can save a lot of resources by reducing the amount of repeated SQL compilations for the same SQL statements. MFC helps performance analysis using MEASURE because dynamically compiled statements are now traceable by their module names. Finally, modules can be modified and recompiled to improve execution plans without the need to change application code.

MFC is a powerful addition to SQL/MX and it must be treated with care. There will be changes in upcoming releases and these will be addressed in a next version of this white paper.

References

The following manuals contain further information. These manuals are located in the NonStop Technical Library section of the NonStop Documentation website at: www.hp.com/go/nonstop-docs.

- *JDBC Type 2 Driver programmer's reference for SQL/MX release 3.x*
- *JDBC Type 4 Driver programmer's reference for SQL/MX release 3.x*
- *SQL/MX connectivity service manual*
- *ODBC/MX Driver for Windows manual*
- *SQL/MX 3.x programming manual for C and COBOL*
- *SQL/MX 3.x reference manual*
- *SQL/MX 3.x query guide*

Please refer to the latest revisions for these manuals.

Appendix: The mxmfc utility script

In addition to "mgscript" that is installed in the mxosrvr installation directory (usually /G/system/zmxodbc/) and can be used to manage modules in the USERMODULES directory, NonStop ATC has developed a utility that can be used to perform routine tasks such as cleaning up module directories, recompile modules and examining execution plans of modules.

The tool is called mxmfc and it provides the following help about the options it supports.

Figure 16: mxmfc help command

```
~> mxmfc -h
mxmfc: valid options are XhceEm:d:lorRsS
--
Version 2.4 Nov 29, 2011
Recompile and manage MFC module files
When compiling module sources with the -c option, the input source is the
generated ".sql" file and must exist.

In addition, the tool can be used to:
- list  execution plan for Query statements
  currently, all statements in a module are listed, although only one is
  the one of interest. The statement name is not always the same
- cleanup the module directory by removing mdf, c, lst and dep files.
  removal preserves the .sql files and module files
Supported options are:

h = This help
d = use module_directory  currently set to: /home/frans
c = invoke SQL compilers to create a new plan
m = used module name        <module name>[.sql]
    Note: the module name should not contain any directory path name
          unless used with the -S (select statement text for a single module)
e = invoke mxci to list execution plan
E = invoke mxci to list DETAILED execution plans (version 2.3.4 and later)
l = invoke mxci to list execution plan
o = use DISPLAY_EXPLAIN on SQL/MX older than version 2.3.4
r = remove unwanted files (.lst .mdf .c .dep )
R = remove unwanted files followed by .sql and module file
S = show SQL DML text for one module ()
    Note: do not specify ".sql", since this is added to the filename by mxmfc
    Note: Wild cards are allowed here. mxmfc locates the filenames using '/bin/head .sql'
s = show SQL DML text from all .sql files
X - disclaimer
```

The tool is available free of charge from HP and is provided "as-is", without warranty. Use the –X option to display the disclaimer.

© Copyright 2012 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. UNIX is a registered trademark of The Open Group.

4AA3-8922ENW, Created February 2012